FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Comparison of Some Parallel Solvers for the Real Full Symmetric Eigenproblem on CRAY T3E

Inge Gutheil

FZJ-ZAM-IB-2000-14

Juli 2000

(letzte Änderung: 18.07.2000)

Preprint: Proceedings of the Sixth European SGI/Cray Workshop, 7-8 September 2000, Manchester, UK

Comparison of Some Parallel Solvers for the Real Full Symmetric Eigenproblem on CRAY T3E

I. Gutheil

John von Neumann-Institut für Computing (NIC)
Zentralinstitut für Angewandte Mathematik
Forschungszentrum Jülich GmbH
July 18, 2000

Abstract

The computation of the eigenvalues and eigenvectors of a full symmetric matrix is a time-consuming subtask in many MPP application codes, especially in the field of quantum chemistry. Several solvers for this problem are available either in public domain or in commercial libraries. Performance, user-friendliness and flexibility are important factors for the acceptance of those libraries even if the routines will only be used as building blocks to develop packages for the solution of specific problems.
The freely available ScaLAPACK [1] library containing two different eigensolvers (PSSYEVX and PSSYEV), the Global Arrays toolkit [3], using an eigensolver from the PeIGS library [6], and the NAG Parallel Library [4] (Jacobi eigensolver and interface to ScaLAPACK) are compared.
The main factors that influence MPP-performance are analysed: single node performance, blocking, scalability, load-balance, and communication overhead. The different routines are compared with respect to these criteria to give the potential user some advice on the possibility to influence the performance and the efficiency of the routines.

1 Introduction

Although symmetric multiprocessor architectures (SMP-systems) will play an increasing role in scientific computing there always will be problems which are too large for this kind of computers in the near future. Thus, there still is need for libraries that support the message-passing programming paradigm.
The solution of the symmetric eigenproblem is a time-consuming subtask in many large MPP application codes, especially in the field of quantum chemistry. There are several solvers for this problem in public domain or in commercial libraries. Performance, user-friendliness, and flexibility are important factors for the acceptance of those libraries.
To help the potential user four different routines for the solution of the real symmetric eigenproblem from three libraries are compared under these aspects.

2 Libraries and Routines Investigated

2.1 Libraries

In general there are three kinds of libraries, public domain libraries, developed mostly by universities and research centres, vendor supplied libraries, and commercial ones.
The first ones are freely available but the user normally has to compile and install them on his own, which sometimes leads to first problems. Support and documentation are usually good but not as comprehensive as for commercial libraries.
Vendor supplied libraries often come with the computer and are included in the machine price. They usually contain basic numerical routines like the BLAS optimised for the specific hardware.
Commercial libraries on the other hand deliver good support, ready-made libraries for a large number of platforms, and extensive documentation, but they are often too expensive for universities or research centres.
We investigated two public domain libraries, ScaLAPACK 1.6 and Global Arrays 2.4 and one commercial library, NAG Parallel Library.
Parts of ScaLAPACK are contained in Cray Scientific Libraries 3.0 (libsci) [2] which is delivered as basic numerical software together with the CRAY T3E. The Basic Linear Algebra Communication Subroutines BLACS [9], which are used for ScaLAPACK's communication, are also part of libsci. Unfortunately, there is a bug in the BLACS library of libsci which leads to problems when subgrids of the original computational grid are created. Thus for all routines requiring redistribution of matrices (i.e. routines from the REDIST library, which is part of ScaLAPACK, are called), the public domain version of the BLACS has to be installed.
We studied Global Arrays 2.4 with PeIGS library version 2. At the time we started our measurements there was no public domain CRAY T3E version of PeIGS available, but as there is a PeIGS library version 2 contained in NWChem, which is installed here, we used this in combination with Global Arrays.

NAG Parallel Library Release 2 was inspected, which we installed with Parallel BLAS (PBLAS) and BLACS from libsci.

2.2 Routines

For the solution of the real symmetric eigenproblem we inspected four routines:

- `PSSYEVX` and `PSSYEV` from the public domain library ScaLAPACK
- `GA_DIAG_STD` from public domain Global Arrays using `PDSPEV` from the also freely available PeIGS library and
- `F02FQFP` from the commercial NAG Parallel Library

`PSSYEVX` is also contained in libsci but we could not find any difference in performance to the public domain version. `PSSYEV` calls routines from REDIST and thus is not contained in libsci.
The routines from the public domain libraries are so-called direct solvers which compute eigenvalues and eigenvectors of a real symmetric matrix in the following three steps:

1. Reduction of the full matrix to tridiagonal form
2. Computation of eigenvalues and eigenvectors of the tridiagonal matrix
3. Back transformation of the eigenvectors of the tridiagonal matrix to those of the original matrix

In all three routines the reduction step is done using Householder transformations and the third step is done applying the Householder matrices of the first step to the eigenvectors of the tridiagonal matrix.
For the second step `PSSYEVX` and `GA_DIAG_STD` use bisection and inverse iteration. If there are clusters of eigenvalues the eigenvectors belonging to these eigenvalues have to be reorthogonalised with this method. In `PSSYEVX` this is done on one single node for each cluster whereas in `GA_DIAG_STD` this is done in parallel.
`PSSYEV` uses a modified QR algorithm for the computation of eigenvalues and eigenvectors of the real symmetric tridiagonal matrix. This guarantees orthogonality of eigenvectors even for clustered eigenvalues. The computation of eigenvalues is not parallelised but computed redundantely on all processors. If eigenvalues are not clustered this leads to about twice as many operations for the solution of the eigenproblem with the QR-algorithm as with bisection and inverse iteration.
The NAG routine `F02FQFP` calculates all eigenvalues and eigenvectors based on a one-sided Jacobi method. A sequence of rotations is applied, each of which zeroes one off-diagonal entry, bringing the matrix to diagonal form. Parallelism is introduced by applying several rotations simultaneously. The one-sided Jacobi method needs more operations but less communication than the two-sided, thus it

was choosen for the parallel implementation. The Jacobi method shows good accuracy and scalability, but is slow, if the matrix is not strongly diagonally dominant. Therefore, the algorithm is not competitive in the general case.

3 User Interfaces

ScaLAPACK and NAG Parallel Library are classical Fortran 77 subroutine libraries whereas Global Arrays supports a global view of distributed data objects and supplies the user with routines to fill those objects with data and manipulate them.

ScaLAPACK as a parallel successor of LAPACK [8] attempts to leave the calling sequence of the subroutines as much as possible unchanged in comparison to the corresponding sequential subroutines from LAPACK. Therefore, ScaLAPACK uses so-called descriptors, which are integer arrays containing all necessary information about the distribution of a matrix. This descriptor appears in the calling sequence of the parallel routine instead of the leading dimension of the matrix in the sequential one. For example the sequential driver routine SSYEVX for the solution of the full real symmetric eigenvalue problem has the following calling sequence:

```
CALL SSYEVX(JOBZ,RANGE,UPLO,N,A(1,1),LDA,VL,VU, &
           IL,IU,ABSTOL,M,W,Z(1,1),LDZ,WORK,    &
           LWORK,IWORK,IFAIL,INFO)
```

whereas the ScaLAPACK routine PSSYEVX is called

```
CALL PSSYEVX(JOBZ,RANGE,UPLO,N,A,1,1,DESCA,VL,VU,   &
           IL,IU,ABSTOL,M,NZ,W,ORFAC,Z,1,1,DESCZ,WORK, &
           LWORK,IWORK,LIWORK,ICLUSTR,GAP,INFO)
```

The data are distributed to a two-dimensional processor grid in a block-cyclic two-dimensional way. The user can choose the block-size and the grid shape for the distribution. He has to care for the correct filling of the local parts of the matrices on his own. There are some auxiliary routines in the TOOLS sublibrary - documented in the Fortran source code files - to help the user to put the right local parts of the matrix to the right places. Additionally, there are example programs available calling PSSYEV and PSSYEVX. These examples do not help the user to determine the minimum sizes of the local parts of the arrays and they only show how to use TOOLS routines to fill local parts of the matrix. For large problems the Users' Guide delivers formulas how to calculate an upper bound for the local dimensions and how to compute local from global indices and vice versa without the TOOLS subroutines.

The usage of Global Arrays is relatively simple although the documentation is not very extensive. The user has a global view of the distributed data. There is no need for transforming global to local indices.

To use Global Arrays the user has to start a memory allocator library with sufficient sizes for MA_STACK and MA_HEAP to allocate memory where Global

Arrays stores the local parts of global data and all workspaces.
The calling sequences for the routines in Global Arrays are very simple because all information about the distribution of data is contained in the global data object and all workspace needed is taken from the memory allocator. Thus the calling sequence for GA_DIAG_STD is the following:

```
CALL GA_DIAG_STD(g_a,g_c,EVALS)
```

where g_a contains the global symmetric input matrix, g_c returns the global eigenvector matrix and EVALS is a local array which on each node contains all computed eigenvalues.
NAG Parallel Library offers routines to generate and distribute matrices and vectors for use in NAG and ScaLAPACK routines, to read and write distributed data objects, and to determine for example the length of workspace needed in ScaLAPACK routines. With the help of these routines the usage of ScaLAPACK can be made easier.
Whereas PSSYEVX has the same flexibility as SSYEVX, i.e. it allows to compute all or only selected eigenvalues and optionally the corresponding eigenvectors, all the other routines calculate the whole eigenspectrum. PSSYEV has the option to compute eigenvalues only, GA_DIAG_STD and F02FQFP always deliver eigenvectors, too.

4 Performance

For this section the letter n will be used to indicate matrix sizes, i.e. $n = 400$ means that the eigenvalues and eigenvectors of a 400×400 matrix are computed. The letter np means the number of processors and nb the block size for the routines PSSYEVX and PSSYEV.
The measurements were restricted to the calculation of all eigenvalues and all eigenvectors of a full real symmetric matrix. We tried to test examples of many different sizes and to study a couple of processor grids and block sizes to find out as many influencing factors as possible. Additionally, we constructed test-matrices with equally spread eigenvalues (no large cluster of eigenvalues) and with one large cluster of $n - 267$ almost equal eigenvalues. The details about the construction of test matrices and the measurements can be found in an internal report of Research Centre Jülich [13].
In the following subsections we will distinguish between "small" and "large" problems, where small and large is related to the number of matrix elements per processor. So a small problem means that each processor has about 200×200 elements of the matrix whose eigenvalues and eigenvectors are to be computed and a large problem means that each processor has at least about 1000×1000 matrix elements. Table 1 shows the actual matrix dimensions of small and the minimal dimensions of large problems on different numbers of processors.

Table 1: Sizes of small and large problems on different numbers of processors

	4 nodes	8 nodes	16 nodes	25 nodes	32 nodes	36 nodes	64 nodes
small problem	$n =$ 400	$n =$ 512	$n =$ 800	$n =$ 1000	$n =$ 1152	$n =$ 1200	$n =$ 1600
large problem	$n =$ 2000	$n =$ 2816	$n =$ 4000	$n =$ 5000	$n =$ 5632	$n =$ 6000	$n =$ 8000

We measured execution times with the intrinsic function `system_clock`. For the analysis of BLAS usage, load balance and commmunication overhead we used the performance analysis tools PAT [10] from Cray Research Inc. and for counting floating point operations we used the performance counter library PCL [12].

4.1 Factors that Influence MPP Performance

There are many factors that influence the performance on MPP systems, some of them - like usage of high-level BLAS routines and blocking strategies - also influence the performance on other machines, some are specific for MPP systems.

4.1.1 Usage of BLAS Routines

All library routines examined use BLAS routines for single node computations, hence vendor optimised BLAS routines, here those from libsci, are a very important factor to improve performance. Due to the small level 1 cache on CRAY T3E for all but very small problems performance of BLAS 1 routines is very poor and even BLAS 2 (matrix-vector) routines cannot deliver high performance. Thus, it is preferable to use BLAS 3 (matrix-matrix) routines because cache may be reused effectively in that case.
From Table 2 it can be seen that the ScaLAPACK routines use blocked algorithms to allow the usage of BLAS 3 routines, whereas `GA_DIAG_STD` and the Jacobi solver from NAG Parallel Library rely only on BLAS 1 routines. `PSSYEVX` is best optimised for BLAS 3 usage if there are no large clusters of eigenvalues. The high percent value of BLAS 1 `SROT` usage in `PSSYEV` is due to the QR-algorithm to compute the eigenvalues of the tridiagonal matrix.
If there is one large cluster of eigenvalues the BLAS 3 and BLAS 2 usage for `PSSYEV` increases slightly wheras BLAS 1 usage decreases. For `GA_DIAG_STD` the overall BLAS 1 usage remains almost the same with one large cluster of eigenvalues. `F02FQFP` shows significant differences in the kind of BLAS 1 routines used: Whereas in the case of no large cluster of eigenvalues most BLAS usage is due to `SROT` and `SDOT` - the computational routines - and only a small part is due to `SSWAP`, in the case of one large cluster of eigenvalues and a small matrix up to 20 % of the time is spent in `SSWAP` and sometimes even more than 50 % `SSWAP` is reached with large matrices.

Table 2: Percentage of time spent in different BLAS routines: no large clusters of eigenvalues. The ranges in percentage arise from different numbers of nodes. For large problems $\geq$ means that for larger problems the percentage is still higher.

percentage of time spent in		small problem	large problem
PSSYEVX	BLAS 3 SGEMM	6 - 8%	$\geq$ 30 %
	BLAS 2 SGEMV	7 - 8 %	$\approx$ 27 %
	BLAS 1	-	$\leq$ 1 %
PSSYEV	BLAS 3 SGEMM	3 - 4 %	$\geq$ 9 %
	BLAS 2 SGEMV	2 - 5 %	7 - 9 %
	BLAS 1 SROT	26-33 %	$\geq$ 56 %
GA_DIAG_STD	BLAS 2, 3	-	-
	BLAS 1 SDOT	28-41 %	39-58 %
	BLAS 1 SAXPY	7 -8 %	19-27 %
F02FQFP	BLAS 2, 3	-	-
	BLAS 1 SROT+SDOT	$\approx$ 64 %	$\approx$ 70 %
	BLAS 1 SSWAP	9-14 %	12-17 %

The most extreme differences between the cases with and without one large cluster of eigenvalues arise with PSSYEVX, but this is only due to the reorthogonalisation of eigenvectors which is done sequentially on one node. This leads to a high BLAS 1 usage of 21 % and only 5 % of BLAS 2 and 5 % of BLAS 3 usage on 4 nodes. The rest of the time is almost completely spent in waits of all nodes not having to reorthogonalise the eigenvectors of the large cluster. Problems with large local parts of the matrices could only be measured on 4 nodes, i.e. with $n = 2000$. For $n > 2500$ all eigenvectors belonging to the large cluster of eigenvalues did not fit into the 128 MB of memory of one node.

4.1.2 Load Balance and Communication Overhead

The communication overhead is another important factor for MPP performance. Problems must not be too small for a larger number of nodes because more nodes usually mean more communication and less computation per node. Table 3 shows clearly that a larger problem per node leads to a smaller percentage of communication.

Load imbalance leads to a high communication overhead as a lot of time is spent in waits for other processors to finish computation and send data needed to continue. From Table 4 it can be seen that the ScaLAPACK routines have better balanced operation counts than GA_DIAG_STD in the case with no clusters. With large matrices this imbalance is extreme. Whereas in both ScaLAPACK routines the difference between least and most operations is less than 10 % of the operation count of the node with least work, in GA_DIAG_STD the node with most operations has up to 70 % more operations to do than the one with least operations.

In the case of one large cluster of eigenvalues load balance remains almost the same for PSSYEV. For GA_DIAG_STD it becomes more imbalanced as reorthog-

Table 3: Percentage of time spent in communication routines: no large clusters of eigenvalues. The smallest values come from 4 nodes and a good matrix distribution, the largest values mostly come from 64 nodes.

percentage of time spent in comm/wait	small problem	large problem
PSSYEVX	28-55 %	6 - 18 %
PSSYEV	19-36 %	2 - 11 %
GA_DIAG_STD	34 - 44 %	14-25 %
F02FQFP	13 - 24 %	8 - 14 %

Table 4: Millions of floating point operations and MFLOPS per node: no large clusters of eigenvalues. The operation counts are the lowest and the highest value per node as delivered by PCL, the MFLOPS are computed by the times measured and these operation counts. Only the highest value for the MFLOPS/node is shown. On the other nodes MFLOPS are lower mainly because of waits.

Million operations per node (MFLOPS per node)		small problem	large problem
PSSYEVX	4 nodes	82-92 (72)	8190-8380 (182)
	32 nodes	219-245 (61)	22000-22700 (182)
	64 nodes	268-331 (56)	30700-32100 (183)
PSSYEV	4 nodes	175-184 (80)	16800-17000 (129)
	32 nodes	455-631 (68)	50400-54200 (133)
	64 nodes	674-945 (65)	74600-81100 (127)
GA_DIAG_STD	4 nodes	124-186 (103)	14500-22000 (117)
	32 nodes	349-585 (76)	37600-64800 (80)
	64 nodes	494-768 (61)	54300-92700 (75)
F02FQFP	4 nodes	750-778 (61)	101000-105000 (54)
	32 nodes	2430-2640 (47)	292000-317000 (48)
	64 nodes	3220-3570 (47)	not measured

onalisation plays an important role.
The most extreme example for load imbalance is the reorthogonalisation of eigenvectors belonging to a large cluster of eigenvalues which is done sequentially on one single node in PSSYEVX. There it can be seen that with 64 nodes, a problem size of $n = 1600$, and a cluster of 1333 eigenvalues about 94 % of the execution time summed up over all nodes is spent in communication/wait. 62 of the 64 nodes only have to execute about 260-320 MFLOP, one node about 560 MFLOP (orthogonalisation of the eigenvectors belonging to one smaller cluster), and one node has to perform about 21900 MFLOP.

4.1.3 Matrix Distribution

ScaLAPACK distributes the matrices in a block-cyclic 2-dimensional way with the block sizes playing an important role for the performance. For detailed information about the data distribution see the ScaLAPACK Users' Guide. It also allows the user to choose the shape of the processor grid, e.g. 2×4 or 4×2 processors, which sometimes influences performance. Usually a square grid gives best performance, but for non square grids we could not always find the optimal grid shape for a given number of nodes.
Global Arrays on the other hand uses the simple distribution to contiguous blocks and redistributes the matrices to the storage scheme which is used in PeIGS. This is a one-dimensional distribution where complete columns are assigned to each node. Since the redistribution is done in the routine `GA_DIAG_STD`, the user has no influence on the assignment of columns to processors.
The one-dimensional distribution leads to an inferior load balance (e.g. only one node computes the complete Householder vector and all the others are waiting) and more one-to-all broadcasting than the two-dimensional distribution where often communication only within one processor row or column is necessary.

4.1.4 Block Sizes

Only for the routines from ScaLAPACK the user can choose block sizes, therefore everything said now only concerns `PSSYEV` and `PSSYEVX`.
Block sizes influence the number of communication steps on one side: large blocks mean long computation steps and few steps of sending large data packets, small blocks mean short computation steps and many steps of sending small data packets. Depending on the machine characteristics and the possibility to overlap communication with computation the optimal blocking has to be found.
On the other hand block sizes also influence load balance:

1. Large blocks mean that at an early stage of the reduction phase processors begin to idle as their local part of the matrix is already tridiagonalised.

2. If $\frac{n}{nb\sqrt{np}}$ is integer for one block size and is not for the other one, the nodes have local parts of the same size in one case whereas with the other block size the sizes of the local parts differ.

Thus, we tried different block sizes and found that block sizes between $nb = 16$ and $nb = 20$ for most problems delivered the best performance and for some of the largest problems a block size of $nb = 32$ was found to be optimal.
With both routines we got a large degradation in performance on 4 nodes for $n = 2048$ and $nb = 16$, (see Figure 1). We found this phenomenon again on 16 nodes with $n = 4000 - 4160$, on 25 nodes for $n = 4900 - 5200$, on 36 nodes for $n = 6000$ and on 64 nodes for $n = 8000$, always with block size $nb = 32$.

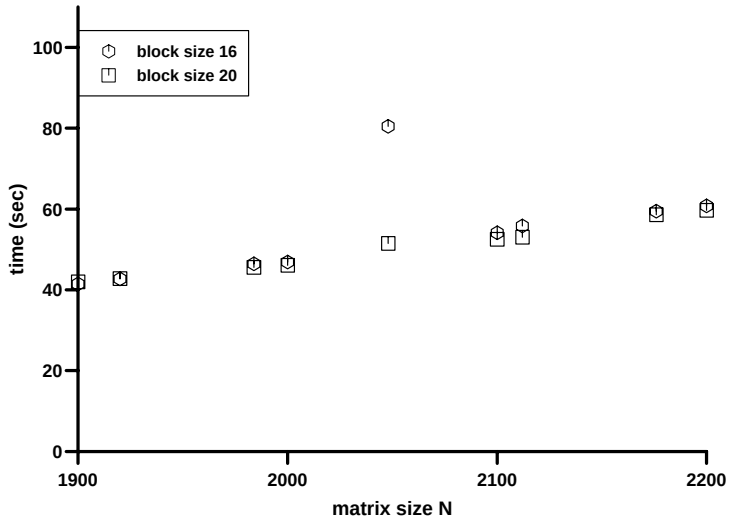

Figure 1: Execution times for PSSYEVX on 4 nodes, different block sizes, computation of all eigenvalues and eigenvectors: no large clusters of eigenvalues

With the help of the performance tool `Apprentice`[11] we could find out that a lot of time was spent in `SGEMM` in the back transformation of the computed eigenvectors. `SGEMM` was called with the second matrix transposed and the first one not, and the matrix size was 1024×1024. This is just the size of at least one of the local matrices in the cases mentioned above.
Performance measurements for `SGEMM` showed that with this size and only in the case with second matrix transposed - first one not - there is a degradation of performance of a factor of about 9. This is due to a performance problem of `SGEMM` from libsci. Called with random matrices the time for `SGEMM` in the above situation is 4.6 sec for $n = 1000$, 54.8 sec for $n = 1024$ and 6.1 sec for $n = 1050$, hence it takes almost 9 times as long to multiply two 1024×1024 matrices - only the second one transposed - than to multiply two 1050×1050 matrices.
With that in mind we suggest not to use block sizes which are powers of two because it is then likely to happen that at least one node has a local block of 1024×1024 matrix elements and the performance slows down significantly due to that problem with `SGEMM`.

4.1.5 Scalability

Figure 2 shows a diagram with speedups for different numbers of nodes compared to four nodes for `PSSYEVX`. Only the case with no large clusters of eigenvalues is shown. In the case of one large cluster of eigenvalues `PSSYEVX` almost does not scale at all, the speedup values are in the range of 1.05 to 1.15 for all numbers of nodes compared to 4 nodes. The scaling of `PSSYEV` and `GA_DIAG_STD` is approximately the same in the case with one large cluster of eigenvalues and without large clusters of eigenvalues.

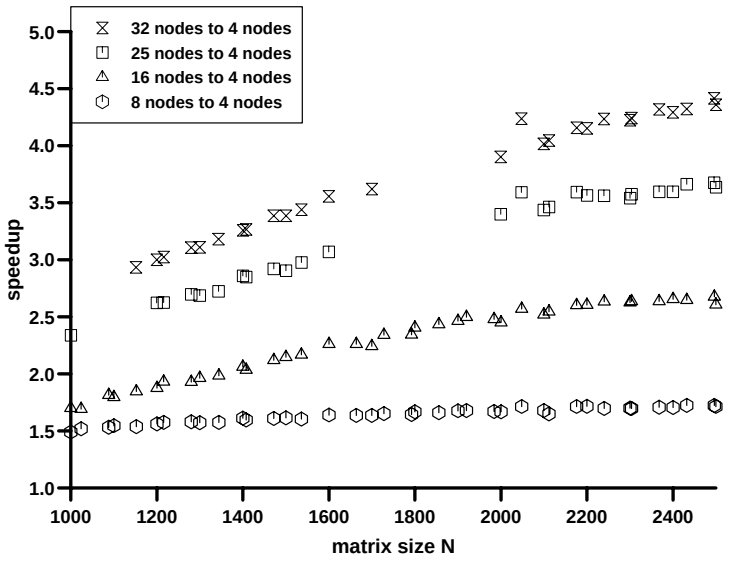

Figure 2: Speedups of PSSYEVX, different numbers of nodes versus 4 nodes, computation of all eigenvalues and eigenvectors: no large clusters of eigenvalues

It can be seen that for 8 nodes the speedup values remain almost constant through the whole range of n, hence the problems are large enough for 8 nodes. For all other numbers of nodes increasing speedup values can be seen.
In Table 5 the maximum speedup values for one number of nodes compared to another one can be seen. For example the speedups of 16 nodes compared to 8 nodes are usually higher than the quotient of the maximum speedup of 16 nodes to 4 nodes and the speedup of 8 nodes to 4 nodes as they are reached at a problem size that is too large to be computed on 4 nodes with 128 MB memory.
Rather good speedup values are reached with all routines if problem sizes are large enough.

Table 5: Maximum speedups reached with different libraries

number of nodes to number of nodes	8:4	16:4	16:8	25:8	32:8	32:16	36:16
PSSYEVX, no large clusters	1.76	2.68	1.63	2.36	2.71	1.79	1.70
PSSYEV, no large clusters	1.85	3.20	1.81	2.53	2.95	1.73	1.90
PSSYEV, one large cluster	1.83	3.15	1.82	2.53	2.92	1.75	1.92
GA_DIAG_STD, no large clusters	1.89	3.49	1.88	2.86	3.51	1.93	2.10
GA_DIAG_STD, one large cluster	1.99	3.67	1.88	2.72	3.45	1.89	2.08

`PSSYEV` scales slightly better than `PSSYEVX`, `GA_DIAG_STD` shows still better scaling but as `GA_DIAG_STD` is slower than the other routines in most cases, the better scalability does not make it superior.

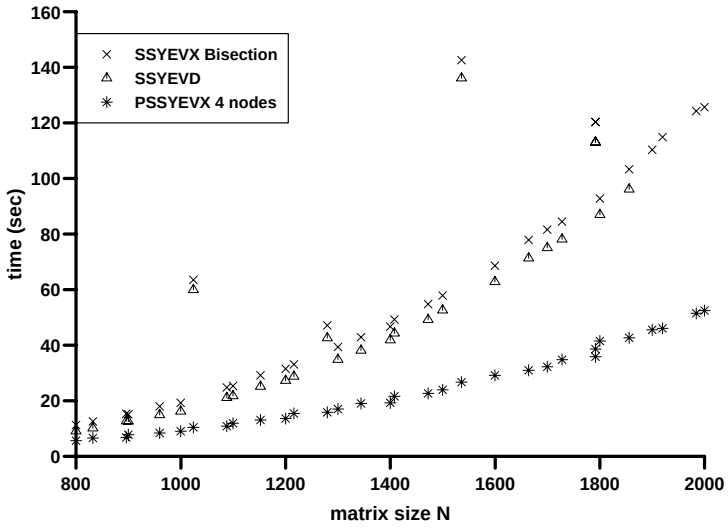

Figure 3: Comparison of the execution times for the best sequential codes with ScaLAPACK codes on 4 nodes, computation of all eigenvalues and eigenvectors on CRAY T3E: no large cluster of eigenvalues

In figure 3 the execution times of PSSYEVX on four nodes in the case with no large clusters of eigenvalues are compared to those of the two best sequential codes, SSYEVD and SSYEVX. SSYEVD uses a divide-and-conquer method for computation of eigenvalues and eigenvectors of the tridiagonal system, SSYEVX uses bisection and inverse iteration just like PSSYEVX.
In this case PSSYEVX on four nodes is about twice as fast as the fastest sequential routine. Here parallelization not only helps to solve larger problems which would not fit into the local memory of one node but it also delivers significant speedup.
In the case of one large cluster of eigenvalues only PSSYEV reaches the execution times of the best sequential code, SSYEVD, already with four nodes and allows larger problems to be solved. Thus, in that case parallelization only helps solving larger problems. If, however, orthogonality of eigenvectors does not have to be very accurate PSSYEVX is faster and allows larger problems, too.

4.2 Comparison of Performance

In the following diagrams of performance results for each routine and each number of processors the configuration (e.g. grid shape, block size) with the shortest execution time is shown.
From figure 4 it can be seen that for equally spread eigenvalues PSSYEVX on 4 nodes is as fast as PSSYEV on 16 nodes. One reason is that PSSYEV needs twice as many operations as PSSYEVX in that case, the other reason is that the sequential QR-algorithm within PSSYEV uses a lot of BLAS 1 routines and consequently reaches less MFLOPS per node than PSSYEVX.

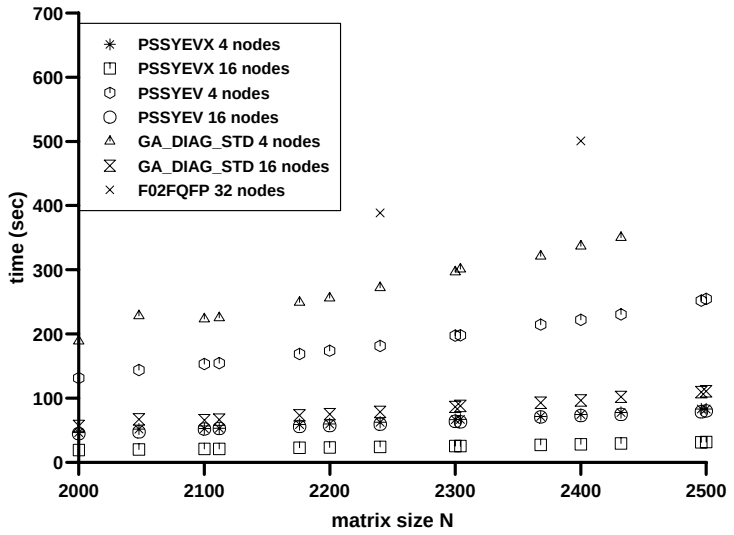

Figure 4: Execution times of the different routines on different numbers of nodes, computation of all eigenvalues and eigenvectors: no large clusters of eigenvalues

Although GA_DIAG_STD uses the same algorithm as PSSYEVX if there are no large clusters of eigenvalues, the number of operations is much higher even for the node with the smallest number of operations (see Table 4). Due to the poor load balance of GA_DIAG_STD it is even higher than that of PSSYEV on the node with the highest operation count. For large problems also the MFLOPS per node reached with GA_DIAG_STD are significantly lower than the ones reached by PSSYEV or PSSYEVX, mainly because it is completely based on BLAS 1 routines and therefore performance is additionally reduced by cache misses.
The Jacobi algorithm F02FQFP of NAG Parallel Library needs about 4 to 5 times as many operations as GA_DIAG_STD or PSSYEV for the general test matrices we used so it cannot be competitive. Nevertheless we show execution times for F02FQFP on 32 nodes. It can be seen that it is slower on 32 nodes than the slowest other routine on 4 nodes.
In the case of one large cluster of eigenvalues the situation changes dramatically. Now PSSYEVX shows the slowest performance and the largest problem that could be solved on four nodes with reorthogonalisation was $n = 2000$. Adding more nodes almost does not speed up execution, it only allows slightly larger problems to be solved.
It can be seen from figure 5 that the execution times for PSSYEV and F02FQFP are slightly lower in the case of one large cluster of eigenvalues than in the case of equally spread eigenvalues. The execution times of GA_DIAG_STD on the other hand become higher as eigenvectors are reorthogonalised. Consequently the difference between PSSYEV and GA_DIAG_STD becomes larger than in the case with no large clusters. F02FQFP on 32 nodes now for small problems is a little faster

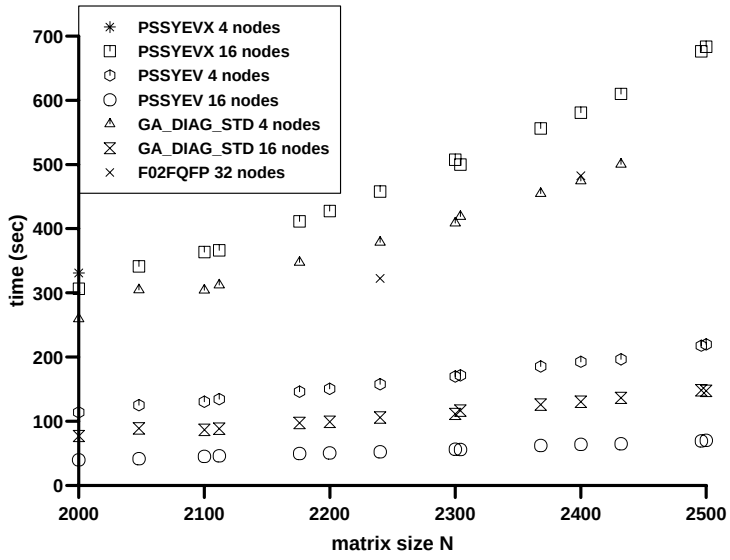

Figure 5: Execution times of the different routines on different numbers of nodes, computation of all eigenvalues and eigenvectors: one large cluster of eigenvalues

than `GA_DIAG_STD` on 4 nodes and `PSSYEVX` on 16 nodes.
For the solution of the symmetric eigenvalue problem there is always one ScaLAPACK routine with highest performance: if eigenvalues are not clustered this is `PSSYEVX`, for one large cluster of eigenvalues whose eigenvectors have to be reorthogonalised `PSSYEV` is the fastest routine.

5 Conclusions and Outlook

ScaLAPACK offers very good performance at the expense of a somewhat complicated user interface. Programmers willing to apply ScaLAPACK routines should become familiar with the data distribution used in ScaLAPACK and adapt their program to this distribution from the start. This will result in good performance and low memory usage.
Global Arrays mainly offers an infrastructure to treat global objects transparently. If routines from other libraries are to be used performance is lost due to the redistribution of data. There is only one interface routine to ScaLAPACK within Global Arrays, the one for LU decomposition and the solution of the resulting triangular system. This routine must be modified if other ScaLAPACK routines like `PSSYEVX` shall be used.
The Jacobi solver contained in NAG Parallel Library is not suited for a general eigenproblem. There are special cases where it may be better than the general purpose routines of ScaLAPACK. NAG Parallel Library contains routines for input and output of matrices distributed in the block cyclic distribution necessary

for ScaLAPACK. This can make use of ScaLAPACK easier.
Libraries like the ones presented here can help developers of new application programs or application packages to take advantage of work already done.
Global Arrays makes parallel programming easy by offering a global view of distributed data, but there are not enough interface routines to other libraries to allow optimal flexibility. With its extensive documentation and example programs for each routine NAG Parallel Library makes parallel programming with ScaLAPACK relatively easy to unexperienced users. Here the problem is that programs using NAG routines are not portable without paying for the implementation of NAG Parallel Library.

In the meantime there exists a new algorithm for the reduction phase in ScaLAPACK, using less communication by combining messages. There is a new public domain version of PSSYEVX available using the new reduction algorithm which may be received via www from **http://www.cs.utk.edu/~kstanley/**. It can be seen that especially for relatively small problems on many nodes (i.e. small matrix parts per node) the performance gain is high if no large clusters of eigenvalues occur. The new reduction algorithm can make the whole computation almost twice as fast on 36 nodes for $n = 400$. The speedup decreases with the problem size and is only about 1.1 for $n = 6000$ on 36 nodes or almost 1.0 for $n = 4800$ on 16 nodes.
If there is one large cluster of eigenvalues and reorthogonalization is necessary, the speedup is very small because the part of the reduction phase in the whole computation is so small in this case.
The new reduction algorithm may be used with PSSYEV as well, but the interface is not ready up to now. The absolute performance gain will be the same but the relative gain, i.e. speedup will be smaller as the computation of eigenvalues and eigenvectors of the tridiagonal matrix takes a larger part of the computation in this case.

In the next release of ScaLAPACK there will be the new reduction algorithm and a parallel version of the divide and conquer algorithm used in SSYEVD. As this was the best performing sequential algorithm this sounds promising. It should be well parallelizable, too.

Since November 1999 Global Arrays 3.0 is available with a new and much better User's Manual. It allows global arrays of up to seven dimensions and contains an additional package for distributed I/O: Disk Resident Arrays. In NWChem now PeIGS 3 is integrated but the eigensolver still is based on BLAS 1 calls. Unfortunately it needs even more operations per node than the one in PeIGS 2 and thus is even slower than the one we investigated.

6 Acknowledgements

I would like to thank Ruth Zimmermann who did the measurements of `F02FQFP` and the sequential codes.

References

[1] L.S. Blackford, J. Choi, A.Cleary et al.
ScaLAPACK Users' Guide
SIAM Philadelphia, 1997
Also available via WWW under
`http://www.netlib.org/scalapack/slug/scalapack_slug.html`

[2] Silicon Graphics Computer Systems / Cray Research
Scientific Libraries Reference Manual, Volume 1-2,
SR-2081 3.0

[3] Global Arrays User Guide
Only available via WWW under
`http://www.emsl.pnl.gov:2080/docs/global/ga.html`

[4] NAG Parallel Library Manual
Release 2
The Numerical Algorithms Group Limited, 1997

[5] NAG Fortran Library - Mark 17
The Numerical Algorithms Group Limited, 1995

[6] D. Elwood, G. Fann, R. Littlefield
Parallel Eigensystem Solver PeIGS Version 2.1, rev. 0.0
Pacific Northwest Laboratory July 28, 1995
Only available via WWW under
`http://www.emsl.pnl.gov:2080/docs/nwchem/doc/peigs/docs/peigs3.html`

[7] Dynamic Memory Allocator Homepage
`http://www.emsl.pnl.gov:2080/docs/parsoft/ma/MAapi.html`

[8] E. Anderson, Z. Bai, C. Bischof et al.
LAPACK Users' Guide, Second Edition,
SIAM Philadelphia, 1995
Also available via WWW under
`http://www.netlib.org/lapack/lug/lapack_lug.html`

[9] Jack J. Dongarra and R. Clint Whaley
A User's Guide to the BLACS v1.1,
`http://www.netlib.org/blacs/lawn94.ps`

[10] J. Galarowicz, B. Mohr
Analyzing Message Passing Programs on the CRAY T3E with PAT and VAMPIR
Forschungszentrum Jülich GmbH, Zentralinstitut für Angewandte Mathematik, Interner Bericht FZJ-ZAM-IB-9809, Mai 1998,
man pat on Cray systems

[11] Cray Research
Introducing the MPP Apprentice Tool, Cray Manual IN-2511, 1994

[12] PCL - The Performance Counter Library
http://www.fz-juelich.de/zam/PCL/

[13] I. Gutheil, R. Zimmermann
Performance of Software for the Full Symmetric Eigenproblem on CRAY T3E and T90 Systems
Forschungszentrum Jülich GmbH, Zentralinstitut für Angewandte Mathematik, Interner Bericht IB-2000-6, July 2000